objects (this is referred to as "child object"), there is a need to prepare a large number of terminals for a relay use for the purpose of connection of objects, when objects to be connected are mutually far hierarchies. Thus, it takes a lot of procedure for a wiring, and thus it is troublesome.

On the other hand, in the event that objects are of a hierarchical structure, and in case of a scheme wherein it is permitted to directly connect objects, which belong mutually different hierarchies, with one another, there will be provided a wiring diagram which does not take into account of a hierarchy. Thus, this raises such a problem that the wiring diagram is not so easy to see and it is difficult to grasp the wiring structure in its entirety.

Further, when there is a need to replace the object once wired by another object, in order to implement the replacement, there is a need to remove the wiring of the previous object and do over again the wiring for the new object. Thus, it takes a lot of procedure for the replacement.

This is a similar as to the matter of that the object once wired on a certain hierarchy is shifted to another hierarchy, for example, a one stage lower-order hierarchy. Also in this case, it takes a lot of procedure such that the wiring of the object before a shift is removed, a parent object is placed wired thereat, the removed object is placed

as a child object of the parent object, and a wiring between the parent object and the child object is conducted.

Further, according to the conventional scheme, there has been associated with such a problem that as the interobject wiring is complicated, a connecting relation among objects is hardly to be understood from an indication of the wiring diagram. Especially, in the event that a bus representative of a flow of request for processing, which bus referred to as a "instruction bus", is connected to a plurality of objects on a branching basis, it is difficult to grasp a running sequence of the processing among the plurality of objects from the indication of the wiring diagram. Accordingly, it is also difficult to alter the running sequence on the wiring diagram.

In view of problems involved in the above-mentioned interobject wiring, the embodiment, which will be described hereinafter, relates to a scheme of facilitating a wiring work.

Fig. 64 is a schematic diagram showing a basic structure of an object-oriented programming supporting apparatus and a program storage medium for use in an object-oriented programming according to an embodiment of the present invention.

An object-oriented programming supporting apparatus

300 supports an object-oriented programming for coupling a plurality of objects each having data and operation with each other in accordance with an instruction. The object-oriented programming supporting apparatus 300 comprises a display means 301, an object coupling means 302, a hierarchical structure construction means 303 and a handler 304.

The display means 301 displays objects each represented by a block representative of a main frame of an object, a data output terminal for transferring data of the object to another object, a data input terminal for receiving data from another object, a message terminal for issuing a message to make a request for processing to another object, and a method terminal for receiving a processing request from another object to execute a method, the object being represented by a hierarchical structure which permits one or a plurality of objects to exist in a single object, and in addition displays a wiring for coupling terminals of a plurality of objects. On the computer system 100 shown in Fig. 1, the display means 301 is constituted of the image display unit 102, a software for displaying the above-mentioned objects and wirings on the display screen 102a of the image display unit 102, and a CPU for executing the software.

The object coupling means 302 constructs a coupling structure among a plurality of objects in accordance with an instruction for coupling terminals of the plurality of objects through a wiring. On the computer system 100 shown in Fig. 1, the object coupling means 302 is constituted of the software for constructing the coupling structure and a CPU for executing the software.

The hierarchical structure construction means 303 constructs a hierarchical structure of objects. On the computer system 100 shown in Fig. 1, the hierarchical structure construction means 303 is constituted of the software for constructing the hierarchical structure and a CPU for executing the software.

The handler 304 instructs a wiring for coupling among objects to the object coupling means 302 in accordance with an operation by an operator (or user), and in addition instructs a position of an object on the hierarchical structure to the hierarchical structure construction means 303. On the computer system 100 shown in Fig. 1, the handler 304 is constituted of the keyboard 103, the mouse 104 and the software for taking in operations of the keyboard 103 and the mouse 104 inside the computer system.

It is noted that the software itself for implementing the object coupling means 302 is also referred to as the

object coupling means, and likewise the software itself for implementing the hierarchical structure construction means 303 is also referred to as the hierarchical structure construction means. A program, in which the object coupling means 302 and the hierarchical structure construction means 303 are combined in the form of software, corresponds to the object-oriented programming program referred to in the present invention. The recording medium 310, in which the object-oriented programming program is stored, corresponds to the program storage medium for use in an object-oriented programming referred to in the present invention. In the computer system 100 shown in Fig. 1, the storage unit 105, in which the object-oriented programming program has been stored, corresponds to the program storage medium for use in an object-oriented programming referred to in the present invention. When the object-oriented programming program is stored in the MO 110, the MO 110 also corresponds to the program storage medium for use in an object-oriented programming referred to in the present invention.

Fig. 65 is a conceptual view showing exemplarily an involving relation among objects. Fig. 66 is a typical illustration showing a connecting relation among objects for defining a hierarchical structure.

As shown in Fig. 65, the whole is considered as one

object, and this is referred to as an object A. The object A includes three objects, that is, an object B, an object C and an object D. The object C includes an object E, an object F and an object G. The object F includes an object H.

If this is expressed with a hierarchical structure, the expression is given as shown in Fig. 66. The hierarchical structure of objects expressed in this manner is referred to as an "object tree".

In Fig. 66, the objects arranged in a horizontal direction implies that they are disposed in the same-order hierarchy. With respect to the objects connected with each other in a vertical direction, the object disposed at higher-order hierarchy implies a parent object, and the object disposed at lower-order hierarchy implies a child object of the parent object.

Fig. 67 is a typical illustration showing a pointer for determining a connecting relation of a certain object to another object.

Each of the objects has, as pointers for defining a parent-child relationship, "pointers to higher/lower-order hierarchy" comprising a "pointer to higher-order hierarchy" and a "pointer to lower-order hierarchy", and as pointers for connecting objects arranged in the same-order hierarchy, "pointers to same-order hierarchy" comprising two pointers of

a "FROM" and a "TO". Further, each of the objects has, pointers for use in wiring representative of a flow of data and instructions among objects, "pointers to buses" comprising two pointers of an "IN" and an "OUT", and "pointers to cables" comprising four pointers of an "instruction", a "data", a "tag instruction" and a "tag data".

The "pointer to higher-order hierarchy" and the "pointer to lower-order hierarchy", which constitute the "pointers to higher/lower-order hierarchy", are, for example, in case of the object A shown in Fig. 66, the pointer to the wiring editor and the pointer to the object B, respectively.

The two pointers of the "FROM" and the "TO", which constitute the "pointers to same-order hierarchy", are, for example, in case of the object C shown in Fig. 66, the pointer to the object B and the pointer to the object D, respectively.

In this manner, there is constructed a hierarchical structure, for example, as shown in Fig. 66, comprising the "pointer to lower-order hierarchy" and the "pointers to same-order hierarchy".

Fig. 68 is a typical illustration showing one of the bus elements constituting the bus element list to be connected to the "pointers to buses" shown in Fig. 67. Fig. 69 is a typical illustration showing one of the cable

elements constituting the cable element list to be connected to the "pointers to cables" shown in Fig. 67. Fig. 70 is a typical illustration showing exemplarily a wiring among objects.

Each of the bus elements arranged on the bus element list defines a bus (terminal) to be connected to another object. Each of the cable elements arranged on the cable element list defines a coupling relation (wiring) between terminals of child object-to-child object when the associated object is given as a parent object.

Fig. 67 shows two pointers "IN" and "OUT" as pointers constituting pointers to the bus. Connected to the pointer "IN" is the bus element list defining a bus which feeds data or messages to the object shown in Fig. 67. Connected to the pointer "OUT" is the bus element list defining a bus which outputs data or messages from the object shown in Fig. 67 toward other object.

In Fig. 67, connected to the pointer "IN" is the bus element list comprising two bus elements BUS 1 and BUS 2. Specifically, the bus element BUS 1 is connected to the pointer "IN", and the bus element BUS 2 is connected to the bus element BUS 1. Connected to the pointer "OUT " is the bus element list comprising two bus elements BUS 3 and BUS 4. Specifically, the bus element BUS 3 is connected to the

pointer "OUT", and the bus element BUS 4 is connected to the bus element BUS 3.

As shown in Fig. 68, each of the bus elements comprises a "pointer to substantial object", "pointer to bus of substantial object", "pointer to next bus element (BUS)" and "other data". It is noted that a terminal of an object is referred to as a "bus".

In the arrangement shown in Fig. 70, in the event that the object shown in Fig. 67 is object A shown in Fig. 70, the bus element BUS 1 corresponds to, for example, "BUS 1" of the object A shown in Fig. 70, and the "pointer to substantial object" corresponds to a pointer to an object (here object B) connected to BUS 1 of object A, of object B and object C included in object A shown in Fig. 70. The "pointer to bus of substantial object" of the bus element BUS 1 corresponds to a pointer to a bus (in case of Fig. 70, BUS 1 of object B) of object B as the substantial object, which bus is connected to "BUS 1" of the object A. The "pointer to next bus element (BUS)" constituting the bus element BUS 1 corresponds, in case of the bus element BUS 1 in Fig. 67, to a pointer to the bus element BUS 2. The "other data" constituting the bus element BUS 1 includes a distinction as to whether the bus (in this case, "BUS 1" of the object A shown in Fig. 70) associated with the bus element is a bus for transfer of data

or a bus for transfer of a message (or instruction). Incidentally, as to an identification between a bus (IN) at the end of receiving data or instruction and a bus (OUT) at the end of transmitting data or instruction, as shown in Fig. 67, it is implemented by separating the "pointers to buses" into "IN" and "OUT".

In Fig. 67, "pointers to cables" comprises four pointers, that is, "instruction", "data", "tag instruction", and "tag data", to each of which a cable element list is connected. Fig. 67 exemplarily shows only a cable element list connected to the "data". Connected to the "data" is directly a cable element CABLE 1. Connected to the cable element CABLE 1 is a cable element CABLE 2. And connected to the cable element CABLE 2 is a cable element CABLE 3.

The "pointers to cables" is used for management of a connecting state (wiring) of buses of child object-to-child object by a parent object. In the example shown in Fig. 70, the wiring of buses between the object B and the object C is managed. Incidentally, the wiring between the object A as a parent object and the object B as a child object, or the wiring between the parent object A and the object C as a child object is managed, as mentioned above, by the bus element list connected to the "pointers to buses".

The four pointers, that is, "instruction", "data",

"tag instruction", and "tag data", which constitute the "pointers to cables", manage a wiring indicative of a flow of messages (instruction), a wiring indicative of a flow of data, a wiring indicative of a flow of an instruction, which is formed dynamically during an execution, as mentioned above, and a wiring indicative of a flow of data, which is formed dynamically during an execution, respectively.

As shown in Fig. 69, a cable element "CABLE" is associated with two terminal elements "TERMINAL". The cable element "CABLE" comprises a pointer to the first terminal element of the two terminal elements "TERMINAL", and a pointer to the next cable element. The terminal element "TERMINAL" comprises a "pointer to an object", a "pointer to a bus of the object", and a "pointer to the next terminal pointer".

Fig. 69 shows exemplarily a cable element for managing a wiring for connecting the bus 2 of the object B with the bus 1 of the object C, shown in Fig. 70, in which the first terminal element stores therein a pointer to an object B and a pointer to a bus 2 of the object B, and the second terminal element stores therein a pointer to an object C and a pointer to a bus 1 of the object C. In this manner, the bus 2 of the object B and the bus 1 of the object C are coupled with each other through the wiring. It is

noted that the first terminal element of the two terminal elements is associated with the bus of the output end of data or instruction, and the second terminal element is associated with the bus of the input end of data or instruction.

The cable element shown in Fig. 69 is managed, as mentioned above, by the object A which is a common parent object for both the objects B and C.

The above are the general explanations of a management of pointers for determining a hierarchical structure of objects, a management of pointers for determining buses of objects, and a management of pointers for determining a wiring for connecting buses of objects. Next, there will be explained more specific embodiments of the object-oriented programming supporting apparatuses according to the present invention, and programs for an object-oriented programming, which are stored in a program storage mediums for use in an object-oriented programming according to the present invention.

According to the first object-oriented programming supporting apparatus of the object-oriented programming supporting apparatuses according to the present invention, and programs for an object-oriented programming, which are stored in the first program storage medium for use in an object-oriented programming, of the program storage mediums

for use in an object-oriented programming according to the present invention, the hierarchical structure construction means 303 shown in Fig. 64 has means for producing a duplicate object of a substantial object designated in accordance with an instruction from the handler 304, and for disposing the duplicate object at a hierarchy different from a hierarchy at which the substantial object is disposed, and the object coupling means 302 receives from the handler 304 an instruction as to a wiring between the duplicate object and another object in the wiring of the hierarchical structure in which the duplicate object is disposed, and constructs a coupling structure in which the duplicate object and the associated substantial object are provided in the form of a united object.

Fig. 71 is a conceptual view of a duplicate object. Fig. 72 is a typical illustration showing a hierarchical structure (object tree) of the objects shown in Fig. 71.

An object A is connected to an wiring editor. Connected to the object A is an object B in a lower-order hierarchy. Connected to the object B is an object C in the same-order hierarchy. Connected to the object C is an object D in a lower-order hierarchy. Connected to the object D is an object E in the same-order hierarchy.

In the event that the objects B and E, which are

disposed at mutually different hierarchy, are connected with each other through a wiring, it is acceptable that a bus (terminal) is formed on the object C which is a parent object of the object E, and the terminal of the object C is connected to the bus of the object E, and in addition the terminal of the object C is connected to the terminal of the object B. However, this work takes a trouble for wiring. In order to avoid such a trouble, according to the present embodiment, a duplicate object E' of which the substantial object is the object E is disposed at the hierarchy at which the objects B and D are disposed, and the bus of the duplicate object E' is connected to the bus of the object B through a wiring on the hierarchy at which the object B and the duplicate object E' are disposed.

Fig. 73 is a flowchart useful for understanding a building process for the duplicate object.

First, in step 73_1, with respect to the designated object (e.g. object E), a duplicate object E' is built through copying the object E. Here a wiring among objects is aimed. Thus, there is no need to copy even the substance of the program constituting the object E and only information necessary for a display and a wiring of objects is copied. In this meaning, the "copy" referred to as the present invention means a copy of information necessary for a display

and a wiring of objects.

Next, in step 73_2, with respect to all buses of the object E,

1. a copy bus (copy bus element) is created on the duplicate object E', and

2. a pointer to the substantial object E and a pointer to the bus associated with the substantial object E, are written.

Fig. 74 is a typical illustration showing a connecting relation between the substantial object (original) and the duplicate object (copy).

Copied on the duplicate object E' are the bus elements BUS 1, BUS 2, ··· arranged in the "pointers to buses" of the substantial object E in the form of an arrangement as it is. Each of the bus elements BUS 1', BUS 2', ··· of the duplicate object E' copied stores a pointer to the substantial object E and a pointer to the associated bus, of the substantial object E (cf. Fig. 68). After the duplicate object is built in this manner, when a wiring between the object B and the duplicate object E' is instructed, as shown in Fig. 71, the associated cable element and two terminal elements are arranged on the "pointers to cables" of the object A which is a parent object of the object B and the duplicate object E' (cf . Fig. 69).

After a wiring work, and when wiring data for interpreter use, which is stored in the wiring data for interpreter use shown in Fig. 2, is generated, the associated bus element of the substantial object E is found from the bus element list of the duplicate object E' to construct an interobject coupling structure in which the duplicate object E' and the substantial object E are formed in a united body as one object.

Next, there will be explained embodiments of the second object-oriented programming supporting apparatus of the object-oriented programming supporting apparatuses according to the present invention, and programs for an object-oriented programming, which are stored in the second program storage medium for use in an object-oriented programming, of the program storage mediums for use in an object-oriented programming according to the present invention.

According to the second object-oriented programming supporting apparatus of the object-oriented programming supporting apparatuses according to the present invention, and programs for an object-oriented programming, which are stored in the second program storage medium for use in an object-oriented programming, of the program storage mediums for use in an object-oriented programming according to the

present invention, the object coupling means 302 shown in Fig. 64 releases a coupling structure of the object before a replacement with another object in accordance with an instruction from the handler 304, and causes the object after the replacement to succeed to the coupling structure of the object before the replacement with another object, and the hierarchical structure construction means 303 disposes the object after the replacement, instead of the object before the replacement, at a hierarchy at which the object before the replacement is disposed.

Fig. 75 is a conceptual view showing a coupling relation of objects before a replacement of objects. Fig. 76 is a typical illustration showing an object tree concerning the objects shown in Fig. 75.

An object A is connected to an wiring editor. Connected to the object A is an object B in a lower-order hierarchy. Connected to the object B is an object C in the same-order hierarchy. Connected to the object C is an object D in the same-order hierarchy. There exists an object E which is not incorporated into the hierarchical structure. The object C is replaced by the object E.

Fig. 77 is a conceptual view showing a coupling relation of objects after a replacement of objects. Fig. 78 is a typical illustration showing a part of the object tree

after a replacement of objects.

When the object C is replaced by the object E, the object E succeeds to the wiring of the object C as it is. Also in the hierarchical structure, the object E is disposed at the hierarchy at which the object C was disposed.

Fig. 79 is a flowchart useful for understanding an object replacing process.

While an interobject network as shown in Fig. 75 is displayed on the display screen 102a (cf. Fig. 1), the mouse 104 is operated to drag an object after replacement (here, the object E) and superimpose the object E on the object C. Where the term "drag" means such an operation that a mouse cursor is placed on the object E displayed on the display screen 102a and a mouse button is depressed, and then a mouse is moved keeping depression of the mouse button. When the the object E is dragged, the object coupling means 302 shown in Fig. 64 identifies that the dragged object is the object E (step 79_2).

When the dragged object E is superimposed on the object C and then dropped, that is, the mouse button is released, in step 79_3, the object coupling means 302 identifies that the object concerned in drop is the object C (step 79_4). In this manner, when it is identified that the dragged object is the object E and the object concerned in

drop is the object C, the object tree is altered from the state shown in Fig. 76 to the state shown in Fig. 78.

This change is implemented in which a manner that, of the pointers of the objects shown in Fig. 76, the pointer to the object E is written, instead of the pointer to the object C, into "TO" of the object B; the pointer to the object B and the object E are written into "FROM" and "TO" of the object E, respectively; and the pointer to the object E is written, instead of the pointer to the object C, into "FROM" of the object D.

Next, the wiring of the object C concerned in drop is retrieved from the cable element list of the object A which is a parent of the object C concerned in drop (step 79_6).

Fig. 80 is a typical illustration showing a part of the cable element list connected to an object A.

It is recorded in this part that the bus 3 of the object C and the bus 4 of the object D are connected to the terminals indicated by the cable element CABLE a. In this manner, the cable elements are sequentially retrieved to identify the wiring connected to the object concerned in drop.

When the wiring connected to the object concerned in drop is identified, as shown in Fig. 80, the wiring is released and connected to the associated bus of the object E after replacement (step 79_7). When the associated bus of

the object E after replacement does not exist and the wire cannot be altered, it is displayed on the display screen 102a and the wiring is cancelled.

Next, there will be explained embodiments of the third object-oriented programming supporting apparatus of the object-oriented programming supporting apparatuses according to the present invention, and programs for an object-oriented programming, which are stored in the third program storage medium for use in an object-oriented programming, of the program storage mediums for use in an object-oriented programming according to the present invention.

According to the third object-oriented programming supporting apparatus of the object-oriented programming supporting apparatuses according to the present invention, and programs for an object-oriented programming, which are stored in the third program storage medium for use in an object-oriented programming, of the program storage mediums for use in an object-oriented programming according to the present invention, the hierarchical structure construction means 303 is in response to an instruction from the handler 304 such that a plurality of objects from among the objects disposed at a predetermined hierarchy are designated and the plurality of objects are rearranged on the lower-order hierarchy by one stage, and rearranges the plurality of

objects on the lower-order hierarchy by one stage, and produces and arranges an object including the plurality of objects on the predetermined hierarchy in such a manner that a coupling structure among the plurality of objects and a coupling structure among the plurality of objects and objects other than the plurality of objects are maintained.

Fig. 81 is a conceptual view showing a coupling relation among objects before a movement of objects. Fig. 82 is a typical illustration showing an object tree concerning the objects shown in Fig. 81.

As shown in Fig. 82, an object A is connected to an wiring editor. Connected to the object A is an object B in a lower-order hierarchy. Connected to the object B is an object C in the same-order hierarchy. Connected to the object C is an object D in the same-order hierarchy. Connected to the object D is an object E in the same-order hierarchy.

It is assumed that the interobject network as shown in Fig. 81 is displayed on the display screen 102a, and the mouse 104 is operated to select the object C and the object D as the objects to be moved to the lower-order hierarchy by one stage.

Fig. 83 is a conceptual view showing a coupling relation of objects after a movement of objects. Fig. 84 is

a typical illustration showing an object tree concerning the objects shown in Fig. 83.

An object F is built on the same hierarchy as that of an object B. An object C and an object E are arranged on a lower-order hierarchy of the object F in the form of children objects of which a parent is the object F.

Before a movement, as shown in Fig. 81, the bus 3 of the object B is directly connected to the bus 1 of the object C. After a movement, however, as shown in Fig. 83, the bus 3 of the object B is connected to the bus 1 of the object F, and the bus 1 of the object F is connected to the bus 1 of the object C. And with respect to a connection of the object D with the object E, the bus 3 of the object D is connected to the bus 2 of the object F, and the bus 2 of the object F is connected to the bus 1 of the object E.

Fig. 85 is a flowchart useful for understanding a processing for a movement of objects and a change of wiring of objects.

When the object, which is to be moved to a lower-order hierarchy by one stage, is selected, it is identified as to what objects (here, objects C and D shown in Fig. 82) have been selected (step 85_1). And a new object (here, object F) is built on the same hierarchy as the selected objects (step 85_2). In step 85_3, the selected

objects (here, objects C and D) are replaced by the new object (object F).

Fig. 86 is a typical illustration showing a state of an alteration of an object tree.

In step 85_2, when the object F is built, the connection between the object B and the object C is cancelled, and the object B is connected to the object F in the same-order hierarchy. And the connection between the object D and the object E is cancelled, and the object F is connected to the object E in the same hierarchy. And the object C is connected to the object F in the lower-order hierarchy. In this manner, the object tree after an object movement, as shown in Fig. 84, is completed.

Incidentally, it is noted that the alternation of the pointer for the alternation of the object tree can be performed in a similar fashion to that of the explanation made referring to Fig. 78, and thus the redundant explanation will be omitted.

Next, as shown in step 85_4 of Fig. 85, the wiring connected to the selected objects (objects B and C) is retrieved from the cable element list connected to the parent object (object A) of the selected objects (objects B and C).

Fig. 87 is a typical illustration showing a part of the cable element list connected to the object A.

In Fig. 87, there are shown that the wiring of the bus 4 of the object C and the bus 1 of the object D are made on the cable element CABLEa, and that the wiring of the bus 3 of the object D and the bus 1 of the object E are made on the cable element CABLEb. Here, it is noted that the wiring of the bus 4 of the object C and the bus 1 of the object D shown on the cable element CABLEa is typically representative of the wiring between the objects (objects B and C) selected to be moved to the lower-order hierarchy by one stage, as shown in Fig. 81, and the wiring of the bus 3 of the object D and the bus 1 of the object E shown on the cable element CABLEb is typically representative of the wiring between the object (object D) to be moved to the lower-order hierarchy by one stage and the object (objects E) not to be moved and to stay at the same-order hierarchy.

In the step 85_4 of Fig. 85, when the retrieval of the cable element list is carried out as mentioned above, the process goes to step 85_5 in which it is determined whether the wiring connected to the selected objects (here objects B and C) located through the retrieval is the wiring between the objects (objects B and C) inside of the new object (object F), or the wiring between the internal object and the external object with respect to the the new object (object F). In this determination, when it is determined

that the wiring of interest is the wiring (corresponding to the wiring of the cable element CABLEa shown in Fig. 87) between the objects inside of the new object (object F), the process goes to step 85_6 in which the wiring is moved from the parent object (object A) to the new object (object F).

Fig. 88 is an explanatory view useful for understanding a movement of wiring to a new object.

The cable element CABLEa is removed from among the cable element list connected to the object A, and is incorporated into the cable element list connected to the object F.

In the step 85_5 of Fig. 85, when it is determined that the wiring of interest is the wiring (corresponding to the wiring of the cable element CABLEb shown in Fig. 87) between the internal object and the external object with respect to the the new object (object F), the process goes to step 85_7 in which a wiring bus is produced on the new object (object F).

Fig. 89 is a typical illustration of a bus for use in wiring, the bus being built on an object F.

In Fig. 89, a bus element BUS 2 is connected to "OUT" (cf. Fig. 67) of the object F. The bus element BUS 2 corresponds to the bus 2 of the object F shown in Fig. 83, and has a pointer to the object D and a pointer to the bus 3

of the object D. That is, the bus element BUS 2 forms, as shown in Fig. 83, a wiring between the bus 2 of the object F and the bus 3 of the object D. It is to be noted that the bus element BUS 2 shown in Fig. 89 is exemplarily shown, and in case of the wiring shown in Fig. 83, a connecting bus element is disposed also in "IN" of the object F so that a wiring between the bus 1 of the object F and the bus 1 of the object C is implemented.

In step 85_8 of Fig. 85, a wiring connected to the object inside a new object (object F) is changed in connection to the new object (object F).

Fig. 90 is a typical illustration showing a state of a change of an object in wiring from an object (object D) inside a new object (object F) to the object F.

The cable element CABLEb of the object A shown in Fig. 87 is indicative of a wiring between the bus 3 of the object D inside the object F and a wiring between the bus 1 of the object E outside the object F. As shown in Fig. 90, the bus 3 of the object D is changed to the bus 2 of the object F thereby forming a wiring between the bus 2 of the object F and the bus 1 of the object E.

Incidentally, the step 85_4 in Fig. 85 is repeatedly performed by a necessary number of times.

Next, there will be explained embodiments of the

fourth object-oriented programming supporting apparatus of the object-oriented programming supporting apparatuses according to the present invention, and programs for an object-oriented programming, which are stored in the fourth program storage medium for use in an object-oriented programming, of the program storage mediums for use in an object-oriented programming according to the present invention.

According to the fourth object-oriented programming supporting apparatus of the object-oriented programming supporting apparatuses according to the present invention, the display means 301 shown in Fig. 64 has, in case of existence of a plurality of method terminals (messages or instructions) connected to one message terminal (a bus for outputting a message or an instruction) designated in accordance with an instruction through the handler 304, means for displaying a list indicative of an execution sequence of a plurality of methods associated with the plurality of method terminals, and the object coupling means 302 has means for reconstructing a coupling structure in which the execution sequence of the plurality of methods appearing at the list displayed on the display means 301 are altered.

Further, according to programs for an object-oriented programming, which are stored in the fourth program storage

medium for use in an object-oriented programming, of the program storage mediums for use in an object-oriented programming according to the present invention, the object coupling means 302 has, in case of existence of a plurality of method terminals connected to a designated one message terminal, means for making up a list indicative of an execution sequence of a plurality of methods associated with the plurality of method terminals, and means for reconstructing a coupling structure in which the execution sequence of the plurality of methods are altered in accordance with an alteration instruction of the execution sequence of the plurality of methods appearing at the list.

Fig. 91 is a typical illustration showing exemplarily a wiring among objects. Fig. 92 is a typical illustration showing a cable element list giving a definition of the wiring shown in Fig. 91.

According to the example shown in Fig. 91, an object A includes an object B, an object C, an object D and an object E. A bus 1 of the object B is connected to a bus 2 of the object C, a bus 2 of the object D and a bus 1 of the object E. Where the bus 1 of the object B serves as a bus (message terminal) for outputting an instruction, and each of the bus 2 of the object C, the bus 2 of the object D and the bus 1 of the object E serves as a bus (method terminal) for

receiving an instruction.

A wiring among these elements is defined, as shown in Fig. 92, by a cable element list connected to the object A (parent object). A number of cable elements are listed on the cable element list shown in Fig. 92. Of those cable elements, a cable element CABLEa defines a wiring between the bus 1 of the object B and the bus 2 of the object C, a cable element CABLEb defines a wiring between the bus 1 of the object B and the bus 2 of the object D, and a cable element CABLEc defines a wiring between the bus 1 of the object B and the bus 1 of the object E.

An instruction (message) outputted from the object B is transmitted to three objects C, D and E in each of which the associated method is executed. In this case, however, it happens that a problem as to an execution sequence among those methods is raised. For example, assuming that the object B serves as an object for inputting data from the exterior, the object C serves as an object for performing an arithmetic operation based on the data inputted, the object D serves as an object for making a graph based on a result of the operation, and the object E serves as an object for displaying the graph, there is a need to execute the respective methods in the order named of the object C, the object D and the object E in accordance with an instruction

indicative of that inputting of the data from the object B is completed.

Here, the wiring shown in Fig. 91 is unclear as to the execution sequence, and consequently, the execution sequence is displayed in the following manner and if necessary the execution sequence is altered.

Fig. 93 is a flowchart useful for understanding processings for a display of an execution sequence for methods and for an alteration of the execution sequence for the methods.

First, for example, while an image as shown in Fig. 91 is displayed, a desired wiring (here, the wiring shown in Fig. 91) is clicked through the mouse 104 to select the wiring of interest. In step 93_1, the object coupling means 302 identifies the selected wiring. In step 93_2, a cable list as to the selected wiring (cable) thus identified is made up and displayed.

Fig. 94 is a typical illustration showing a cable list element list.

When the cable list is made up, a cable element list of the parent object (object A) shown in Fig. 92 is retrieved, the cable elements CABLEa, CABLEb and CABLEc, which constitute the selected wiring, are identified, and pointers to cable elements are stored in cable list elements

constituting the cable list element list shown in Fig. 94 in the order listed in the cable element list. That is, in case of the present example, the pointers to three cable elements CABLEa, CABLEb and CABLEc, which constitute the selected wiring, shown in Fig. 92, are stored in the order named in the respective associated cable list elements arranged in the cable list element list shown in Fig. 94.

Fig. 95 is a view exemplarily showing a cable list displayed on a display screen 102a.

When the cable list element list as shown in Fig. 94 is made up, a state of the respective wiring for coupling two objects with each other is displayed with an arrangement according to the order listed in the cable list element list. Specifically, according to the example shown in Fig. 95, it is displayed on the first line that the bus 1 of the object B is connected to the bus 2 of the object C; it is displayed on the second line that the bus 1 of the object B is connected to the bus 2 of the object D; and it is displayed on the third line that the bus 1 of the object B is connected to the bus 1 of the object E. Where the line is referred to as a "list item". The left side of the cable list denotes a bus of the end for generating an message (instruction), and the right side of the cable list denotes a bus of the end for receiving and executing the message

(instruction) generated. In the practical operation, when the bus 1 of the object B issues the associated message (instruction), the respective methods are executed in accordance with the sequence shown in the cable list.

In step 93_3 in Fig. 93, it is assumed that a line of list item indicated in the display list is dragged. Here it is assumed that the list item "object B : bus 1 object E : bus 1" appearing on the third line of the cable list shown in Fig. 95. In step 93_4, the object coupling means 302 (cf. Fig. 64) identifies that a wiring for connecting the bus 1 of the object B to the bus 1 of the object E, that is, the wiring defined by the cable element CABLEc shown in Fig. 92 is dragged. In step 93_5, the dragged list item is dropped. Where it is assumed that the dragged list item is dropped on the second list item "object B : bus 1 object D : bus 2" of the cable list shown in Fig. 95. In step 93_6, the object coupling means 302 identifies that the wiring concerned in drop is a wiring for connecting the bus 1 of the object B to the bus 2 of the object D, that is, the wiring defined by the cable element CABLEb shown in Fig. 92.

Thus, when the dragged wiring and the wiring concerned in drop are identified, an arrangement sequence or the execution sequence is altered in such a manner that the dragged wiring is arranged before the wiring concerned in

drop on the cable list shown in Fig. 95 (step 93_7).

Fig. 96 is a typical illustration showing a state in which an arrangement sequence of the cable elements arranged on the cable element list is altered. Fig. 97 is a typical illustration showing a cable element list in which an arrangement sequence of the cable elements has been altered.

As shown in Fig. 69, each of the cable elements CABLE has a pointer to the next cable element. Thus, when the drag and drop operations for the list item are performed in the manner as mentioned above, the pointer is rewritten. In this example, as shown in Fig. 96, an arrangement sequence of the cable elements is altered in such a manner that the cable element CABLEc is arranged before the cable element CABLEb, and thus the cable element list, in which the cable elements are arranged as shown in Fig. 97, is made up.

Fig. 98 is a typical illustration showing a state in which an arrangement sequence of the cable list elements arranged on the cable list element list is altered. Fig. 99 is a typical illustration showing a cable list element list in which an arrangement sequence of the cable list elements has been altered.

When the drag and drop operations for the list item are performed in the manner as mentioned above, an

arrangement sequence of the cable elements, in which the cable elements are arranged in the cable element list as shown in Fig. 96, is altered. Following this, an arrangement sequence of the cable list elements, in which the cable list elements are arranged in the cable list element list as shown in Fig. 98, is altered. According to this example, an arrangement sequence of the cable list elements is altered in such a manner that the cable list element storing therein the pointer to the cable element CABLEc is arranged before the cable list element storing therein the pointer to the cable element CABLEb, so that the cable list element list shown in Fig. 99 is made up.

Fig. 100 is a view showing a cable list in which an arrangement sequence has been altered.

As a result of alterations of the arrangement sequences of the cable elements and the cable list elements as mentioned above, the cable list for a display is also altered in a sequence of the list item, as shown in Fig. 100.

The above is an explanation of the embodiments of the interobject wiring editor unit 122 and its periphery. Next, there will be explained an explanation of the embodiments of the object builder unit 121 and its periphery.

The object ware programming system aims to perform an efficient programming through replacing programs by objects.

For this reason, it is very important as to whether the existing software can be readily replaced by an object. Particularly, if it is possible to directly replace the existing software by an object, the number of the available objects is dramatically increased all at once, and as a result, a program development efficiency is extremely improved. Hitherto, there have been proposed several types of methods in which the existing software is replaced by an object. An OLE and a DDE in Windows are raised by way of example. However, according to those methods, it is needed to estimate beforehand at the existing software end that the existing software is replaced by an object. And thus, it is difficult to replace all of the existing softwares by objects. Further, even if the associated existing softwares are concerned, many of those softwares are involved in one which is very few in number of messages to be acceptable as compared with, for example, that of the graphical user interface. Accordingly, it is impossible to handle the existing softwares in a similar fashion to that of the graphical user interface.

With respect to a continuous operation for the existing softwares, hitherto, there is known a method in which a description is performed by the shell script. However, according to the earlier technology, it is difficult

to perform an operation for the software after the actuation in a similar fashion to that of the graphical user interface. Further, with respect to the description of the shell script, it must be performed by a user self and thus it will be difficult for a beginner user poor in experience of a programming to do so.

In view of the problems on building the objects as mentioned above, the embodiments, which will be described hereinafter, relate to a scheme of replacing the existing software by an object independently of types of the existing software, and a component which serves as an object in combination with the existing software. Here, there will be described, with the existing software having the graphical user interface as a main software, a scheme of replacing the existing software by an object, and a component which serves as an object in combination with the existing software.

A corresponding relation between the component described hereinafter and the present invention is as follows.

When the component, which will be described hereinafter, is stored in the storage unit 105 of the computer system 100 shown in Fig. 1, the storage unit 105 storing the component corresponds to one example of the component storage medium referred to in the present invention. In a case where the component is stored in the MO

110 shown in Fig. 1, the MO 110 storing the component corresponds to an alternative example of the component storage medium referred to in the present invention.

Fig. 101 is a typical illustration showing an embodiment of a component "including" an existing software having a graphical user interface.

In Fig. 101, an application A is an existing software in which while icons such as "button 1", "button 2", and "button 3" are displayed on the display screen 102a (cf. Fig. 1), anyone of those icons is clicked through an operation of the mouse so that a processing associated with the clicked icon is executed.

A window management unit manages a graphical user interface of all applications incorporated into the system, including the application A. For instance, if it is a Windows, the window management unit denotes a Windows system itself. A component A "including" the application A has a basic structure as an object, for connecting with other objects, and in addition data related to the application A. The component A has further as a method an application drive program and a window event generation program (e.g. a button 1 click event issue program for executing the equivalence to such a matter that a user clicks the button 1 through an operation of the mouse 104). When a message is

transmitted from another object to an application A drive method of the component A, the method is executed to drive the application A so that information (e.g. ID information and the like) related to the window is read and the component A maintains the window information.

Further, when a message is transmitted from another object (or one's own self) to a method which issues an event such as a button click or the like, the associated event is issued through the window management unit to the window of the application A in accordance with the event issue program described in the method which received the massage.

In this case, it is possible to replace the existing application by an object by means of simply adding the component A, maintaining the existing application A as it is.

Fig. 102 is a typical illustration showing an alternative embodiment of a component "including" an existing software having a graphical user interface.

In the embodiment explained referring to Fig. 101, added to the last of the event generation program for the existing application A, the existing software and the like is a program for issuing a message to inform other object of that an execution of the method is finished. The message thus issued is connected to a method of other component or other object. Thus it is possible to execute a plurality of

methods on a chain basis. In Fig. 102, the existing software is omitted, and there is shown the state that the messages of the component A are connected to the methods of the components B and C.

Fig. 103 is a typical illustration showing a further alternative embodiment of a component "including" an existing software having a graphical user interface.

The component shown in Fig. 103 is an example of a component having such a function that events for the existing software are monitored and when a predetermined event is issued, the associated message is issued.

When a method for driving an application A of a component A being an existing software receives a message, the method is executed to drive the application A. The component A has a function to monitor all window events and investigates as to whether the issued event is involved in the application A. When it is identified that the issued event is involved in the application A, the component A issues a message for informing another object (or one's own self) of the fact that the event was issued for the application A. For example, when the icon "button 1" of three icons "button 1", "button 2" and "button 3" related to the application A, which are displayed on the display screen 102a, is clicked through an operation of the mouse 104 by a

user, the component A identifies that the icon "button 1" of the application A was clicked, and issues a message for informing that the button 1 was clicked.

In this manner, it is possible, upon receipt of an issue of the event of an existing software, to execute on a cooperative basis a method which does not appear on a specification of the existing software, without adding advanced functions to the existing software.

Fig. 104 is a typical illustration showing a structure of an event processing portion of the window management section shown in Fig. 103. Fig. 105 is a typical illustration showing a structure of an event monitor portion of the component A shown in Fig. 103.

The event processing portion of the window management section is a part in which upon receipt of the issue of an event, a processing associated with the event is carried out. The event processing portion has an event processing element list consisting of a plurality of event processing elements each storing therein pointers to various types of event processing functions. When a window event is generated, the event processing functions indicated by the pointers stored in each of the event processing elements are sequentially executed. The event processing element, which is arranged at the last of the event processing element list,

indicates a default event process function. The default event process function serves, for example, when a button is clicked, to perform such a processing that a button on the display screen is moved as if the button on the display screen is depressed.

At the last of a drive method of the application A of the component A show in Fig. 103, there is described a program for requesting the window management unit to transmit the window event to one's own self (component A). Specifically, the event processing element, which stores therein a pointer to an event monitor portion of the component A, is added to the event processing list possessed by the event processing portion of the window management unit. In this manner, it is possible thereafter to refer to the occurred event at the event monitor portion of the component A, whenever the window event occurs.

The event monitor portion of the component A stores an event table shown in Fig. 105 in which described are a window ID for defining events concerning the application A, an event ID, other data, and a message issued when the event issued, in their corresponding relation.

When any of the window events occurs and event data related to the occurred window event is inputted through the window management unit shown in Fig. 104 to the event monitor

portion of the component A, the event table is referred to by the window ID and the event ID of the event data to retrieve as to whether a window ID and an event ID, which match the window ID and the event ID of the event data, respectively, exist in the event table. When it is determined that a window ID and an event ID, which match the window ID and the event ID of the event data, respectively, exist in the event table, the component A issues a message associated with the matched window ID and event ID.

Fig. 106 is a basic construction view of a component builder apparatus according to the present invention.

The component builder apparatus 400 comprises a first handler 401, a second handler 401 and a component builder means 403.

The first handler 401 serves to selectively indicate making of methods and messages.

The second handler 402 serves to input an instruction of an issue of a desired event of a predetermined existing software.

It is to be noted that while the first handler and the second handler are functionally separately distinguished from one another, it is acceptable that these handlers are constructed in form of a united body on a hardware basis. In the computer system shown in Fig. 1, the mouse 104 typically

corresponds to both the first handler and the second handler.

The component builder means 403 builds a component which serves as one object in combination with an existing software. Specifically, the component builder means 403 serves, when making of a method is instructed by an operation of the first handler 401 and a predetermined event of the existing software is issued by an operation of the second handler 402, to make on the component a method which fires with a message issued by another object and issues the event, and serves, when making of a message is instructed by an operation of the first handler 401 and an issue of a predetermined event of the existing software is instructed by an operation of the second handler 402, in response to an occurrence of the event, to make on the component a message for informing other objects of the fact that the event occurred.

The component builder means 403 corresponds to the object builder unit 121 of the object ware programming system 120 shown in Fig. 2.

Fig. 107 is a typical illustration useful for understanding an embodiment of a component builder apparatus according to the present invention. Fig. 108 is a flowchart useful for understanding processings of building a component using a component builder apparatus.

An object builder portion 121 has a program 121a for building a component "including" or "involving" an existing software, which serves as one object together with the existing software. In step 108_1, the existing software (here application A) "included" from the program is driven in accordance with an instruction from a user. In step 108_2, window information of the application A is obtained and maintained.

Next, in step 108_3, the user makes a selection as to whether a method or a message is added to the component "including" the driven application A, and further makes a selection as to types of events (for example, a distinction between the button click and the menu click). The selection between the method and the message mentioned above is carried out in accordance with such a way that either one of the icons of a method and a message on the display screen is clicked by the mouse. A name of the method or the message to be added is registered into the selected column through an operation of the keyboard.

In step 108_4, an occurrence of events is monitored. When an event is generated by the button click or the like using a mouse (step 108_5), it is determined as to whether the generated event relates to a window of the application A (step 108_6). Further, in step 108_7, it is determined as to

whether the generated event is the same type of event as the type (e.g. a distinction between the button click and the menu click) of the event selected in step 108_3.

With respect to the mechanism (functions of the window management unit and the event monitor portion) for determining as to whether the generated event is a desired event, it is the same as that explained referring to Figs. 103 to 105. Thus, the redundant explanation will be omitted.

When the generated event is concerned with the window of the application A and in addition is of the same type as the selected event, the event is added to the component A in the form of the method or the message in accordance with a distinction between the method and the message selected in step 108_3 together with the type of event. In other words, there is added a program such that when a message is received from another object at the component A "involving" the application A, a method of causing the event to generate is created, or when the event is generated, a message, which stands for that the event is generated, is informed to another object.

The above-mentioned operation is continued until a user gives an instruction for termination of monitoring an event (step 108_9). Upon receipt of the event monitoring termination instruction given by the user, the application

(application A) now on drive is terminated in drive. Further, with respect to an object comprising the application A and the component A "involving" the application A, object data for display and wiring as to such an object is created and stored in the object data file 132, and the object is compiled to create running object data and the running object data is stored in the running object file 133. In this manner, the component "involving" a desired existing software is built on an interactive basis.

Next, there will be explained embodiments of the fifth object-oriented programming supporting apparatus of the object-oriented programming supporting apparatuses according to the present invention.

Fig. 109 is a construction view of an object ware programming system in which structural elements corresponding to the embodiment of the fifth object-oriented programming supporting apparatus according to the present invention are added to the object ware programming system 120 shown in Fig. 2. In Fig. 109, the same parts are denoted by the same reference numbers as those of Fig. 2, and the redundant description will be omitted.

An object ware programming system 120' shown in Fig. 109 comprises, in addition to the structural elements of the object ware programming system 120 shown in Fig. 2, an event

log generating unit 141, a component coupling unit 142, an event log file 151 and a component file 152.

According to the embodiments of the component builder apparatus explained referring to Figs. 107 and 108, the built component is stored in the object data file 132 and the running object file 133. On the contrary, according to the present embodiment shown in Fig. 109, while it is the same as the former embodiment with respect to the running object file, data for display and wiring of the built component is stored in the component file 152 instead of the object data file 132. It is to be noted that for the purpose of better understanding, the component file 152 is formed independently of the object data file 132, but it is acceptable that the component file 152 and the object data file 132 are constructed in the form of united body.

First, in accordance with the scheme explained referring to Figs. 107 and 108, upon receipt of a message, an event of an existing software is issued, and a component, which outputs it in the form of a message that the event is issued, is built on each of a plurality of existing softwares and stored in the component file 152.

Next, a user drives simultaneously or sequentially those existing softwares in many number to generate a various types of events. Then, the event log generating unit 141

generates an event log indicative of as to what event is generated in what order. The event log thus generated is stored in the event log file 151.

When a generation of the event log is terminated, the component coupling unit 142 sequentially reads the events stored in the event log file 151 and wires the components stored in the component file 152 so that the events read out are sequentially generated.

A wiring result is stored in the interobject wiring data file 134. Further, if necessary, an additional wiring is conducted by the interobject wiring editor unit 122, and then the wiring is converted into wiring data for an interpreter use and stored in the wiring data file 135 for an interpreter use.

Fig. 110 is a flowchart useful for under standing an operation of a component coupling unit. Fig. 111 is a flowchart useful for understanding an operation of a component coupling unit.

As shown in Fig. 111, th event log file stores therein an event log in which a number of event data are arranged, which is generated in the event log generating unit 141 (cf. Fig. 109). The component file (cf. Fig. 109) stores therein a number of components in which the event is associated with the method in accordance with a manner

mentioned above.

In the component coupling unit, as show in Fig. 110, an event is loaded by one from the event log file (step 110_ 1). In step 110_ 2, the loaded event is compared with a description of a corresponding relation between an event and a method, the description being possessed by a component stored in the component file, and the same event as the loaded event is retrieved from the component file. When the same event is identified, a wiring between a method associated with the event thus identified and a previous message (which will be described below) is conducted (step 110_ 3). A message, which is issued when the method is executed, is saved in the form of the "previous message". Regarding the "previous message", it is noted that the component file stores therein, as shown in Fig. 102, such a type of component that when a method is executed, a message indicative of that an event associated with the method is issued is issued. When the succeeding event remains in the event log stored in the event log file 141 (step 110_ 4), the process returns to step 110_ 1 in which the succeeding event is loaded, and a wiring is conducted in a similar fashion to that of the above.

Incidentally, with respect to the event which is arranged at the first of the event log, no "previous message"

exists. Thus the wiring between a method and the previous message, as shown in Fig. 111, is not conducted, and a message, which is issued when the method issuing the event is executed, is saved in the form of the "previous message".

In this manner, it is possible to implement an automatic wiring among components. This wiring makes it possible in execution by the interpreter unit 123 to automatically sequentially issue events in accordance with the sequence of generation of the event log by a user, and thus an automatic operation for the existing software is possible.

When the event log is once produced, the automatic wiring is conducted sequentially in accordance with the sequence of the events arranged on the produced event log. It is also acceptable, however, that the event log once produced is displayed in the form of a table, and a user selects a necessary event from the table displayed so that an automatic wiring is conducted in accordance with a sequence selected by the user. According to this way, it is possible, when errors occur during a generation of an event log, to correct the errors without doing over again in generation of the event log.

In this manner, it is possible to implement, for example, an autopilot function of the WWW browser, by means

of implementing an automatic operation of the application.

Next, there will be explained an alternative embodiment of a scheme in which an existing software is "involved" and replaced by an object, and an alternative embodiment of a component which serves as an object in combination with an existing software.

Fig. 112 is a conceptual view showing a state in which an existing software is "included" in a component. Fig. 113 is a view showing a table for definition items to give various definitions shown in Fig. 112. In Fig. 113, the object is referred to as "LSI".

Here, the existing software is an existing program consisting of a function or a set of functions, not solely executed but executed when called from other application program or the like.

In the existing programming, there exist data x_1, x_2, x_3, ···, x_i, ··· to be received from other programming, functions function 1, function 2 ···, function j, ··· for performing a processing based on the received data, and data y_1, y_2, ···, y_j, ··· to be transmitted to other program, which are representative of a result of processing.

When such a program is "involved", as shown in Fig. 112, it is assumed that an object is defined with a separation into two parts. The separating way is given with

a certain degree of option, and may be determined by a user.

Here, various types of definitions are given as shown in Fig. 112. First, as (A) a header, there are defined a project name for specifying the whole of works or processings and an environment for executing the processings.

(A) a header is followed by (B) a definition to be made up, (C) a definition of an existing program (defining as to which existing program is to be replaced by an object), and (D) a definition of an object. It is noted that (D) a definition of an object is given with a plurality of definitions when the existing program is partitioned into a plurality of objects.

In (D) a definition of an object, there exist a definition of a data bus (a data input terminal) for use in data input for identifying a pointer which receives data from other object, a definition of a method (a method terminal) for identifying a pointer of an entrance of the processing to be executed, and a definition of a data bus (a data output terminal) for use in data output for identifying a pointer for data to be transmitted to other object. It is to be noted that according to the present embodiment, since the existing program adapted for executing a processing when called from other program is assumed, it is not considered that this existing program requests (an issue of message) of

another object a processing.

Fig. 113 is a view showing a table for definition items to give various definitions shown in Fig. 112.

The keyword groups appearing on the table are of a kind of program language useful for giving the above-mentioned various definitions. A detailed explanation of the individual keywords will be omitted, since it is not essential to the present invention.

Fig. 114 is a view exemplarily showing images displayed on a display screen 102a when definitions are given.

The left side of the screen shows structures of definitions, each of which serves as an icon. When any of the icons is clicked, there is displayed as shown at the right side of the screen a frame of a table for giving a definition of the item associated with the clicked icon. Filling the frames one by one completes a definition table.

An adoption of such a type of scheme that the frames of the table is filled makes it possible to readily give a definition on an interactive basis.

When the definition table is completed, an existing program and a component comprising the definition table related to the existing program are stored in the object data file 132 with an extraction of data for display and wiring by the object builder unit 121 shown in Fig. 2, and also are

stored in the running object file 133 through a conversion into a running format by a compiler.

In this manner, it is possible to take in an existing software to the object ware programming system in the form of the object, regardless of a format of the existing software, maintaining the existing software as it is.

As described above, according to the present invention, it is possible to specially enhance reuse of the software, and also to implement the software higher in the running speed.

While the present invention has been described with reference to the particular illustrative embodiments, it is not to be restricted by those embodiments but only by the appended claims. It is to be appreciated that those skilled in the art can change or modify the embodiments without departing from the scope and spirit of the present invention.